Building complicated functions

(Hints for polynomial interpolation)

Dr. Sean Laverty

10/9/2019

The following tutorial is meant to help you think about ways to streamline the computational definition of complicated functions in R. Appending terms is fairly complicated for arbitrary x. Since x is unknown and R is natively numeric and not symbolic complications arise. Below are some strategies to cope with this potential limitation. Other approaches include using the Ryacas package or switching to Mathematica or other languages.

Product-defined functions

Say we wanted to build a function as follows

$$f(x) = \prod_{i=0}^{n}(x - x_i)$$

or

$$g(x) = \sum_{i=0}^{n}(x - x_i).$$

The implementation is surprisingly intricate.

```
(xs <- seq(1, 10, by=1))
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Perhaps we want to use the first three of these points, for simplicity, taking $n = 2$.

```
fmanual <- function(x) (x-xs[1])*(x-xs[2])*(x-xs[3])
```

This works for a small number of terms. but beyond this gets quite tedious and hard to maintain.

```
f <- function(x, n){
  val <- 1
  for(i in 1:n){
    ## what would happen if we wanted to exclude a certain i from the product?
    val <- val*(x - xs[[i]])
  }
  return(val)
}
```

Comparing values for three-term polynomials we have the following.

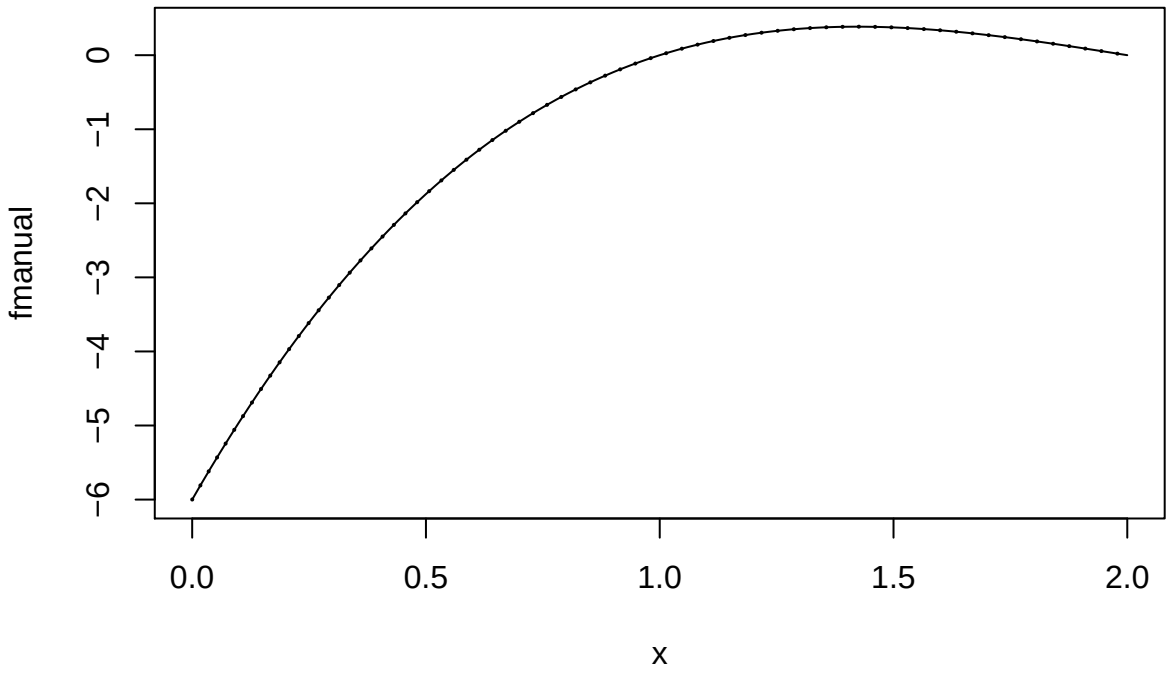
```
fmanual(1)
```

```
## [1] 0
```

```
f(1, 3)
```

```
## [1] 0
```

```
plot(fmanual, xlim=c(0, 2))
plot(function(x)f(x,3), xlim=c(0, 2), add=T, lty=3, lwd=2)
```




```
## graphs are superimposed
```

More importantly the second construction extends to an arbitrary number of factors quite easily (though we don't have access to the coeeficients of the expanded form of the polynomial). In the plot, colors indicate increasing orders of the polynomial from

$$f(x;1) = (x - x_0) = (x - 1)$$

to

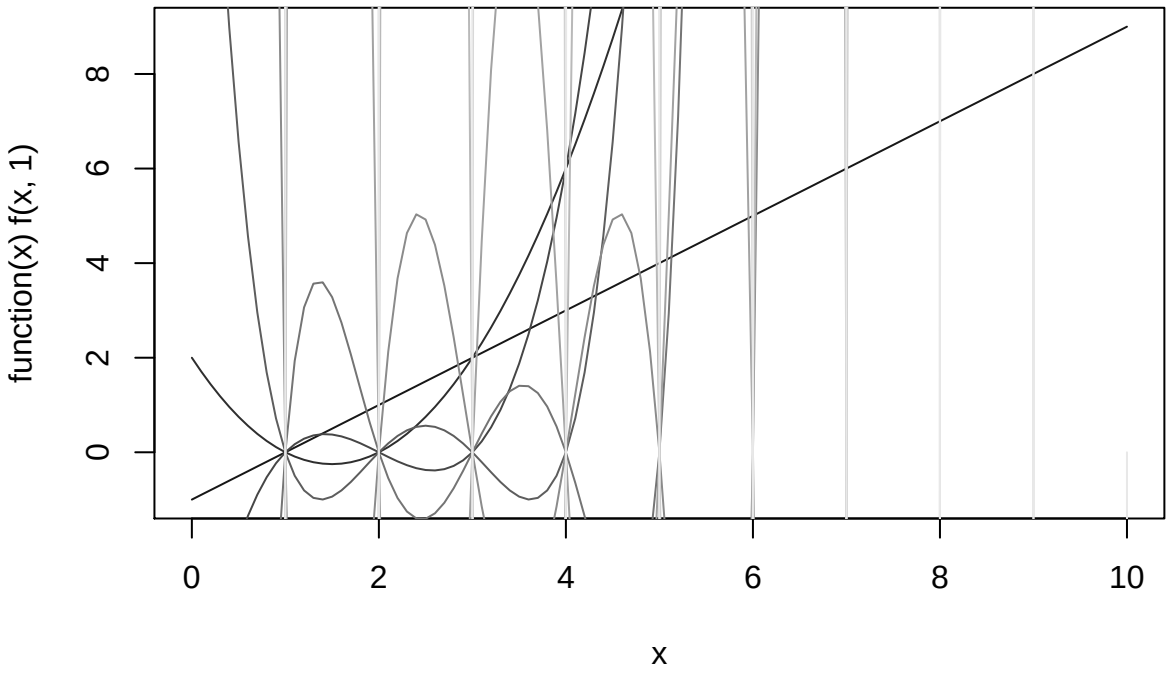
$$f(x;2) = (x - x_0)(x - x_1) = (x - 1)(x - 2)$$

through

$$f(x;9) = (x - x_0) \cdots (x - x_9) = (x - 1) \cdots (x - 10).$$

Remeber the indexing is off by one $x_0 = 1$ (coded as `xs[1]`).

```
plot(function(x)f(x, 1), xlim=c(0, 10), col=gray(1/(length(xs)+1)))
## '+1' just ensures we don't have a color that is pure white
for(i in 2:length(xs)){
  plot(function(x)f(x, i), xlim=c(0, 10), add=T, col=gray(i/(length(xs)+1)))
}
```



Summation-defined functions

Extend this code to include a function defined as a sum.

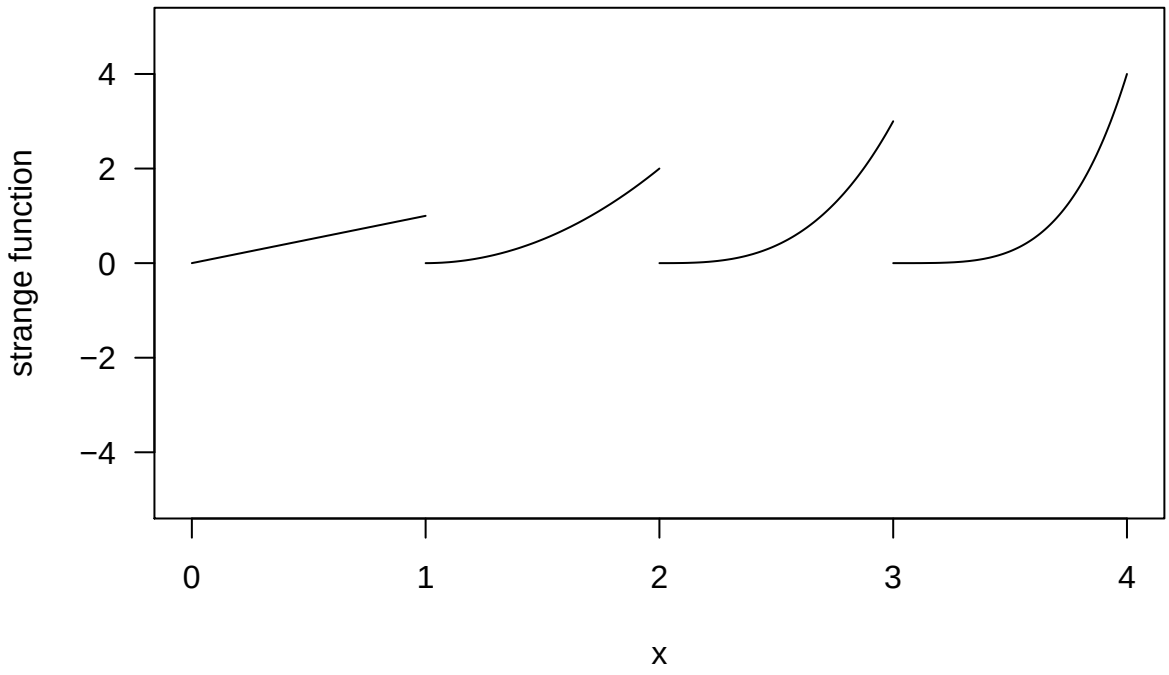
Piecewise functions

Consider the piecewise-defined function

$$f(x) = \begin{cases} 1(x-0)^1, 0 \le x < 1 \\ 2(x-1)^2, 1 \le x < 2 \\ 3(x-2)^3, 2 \le x < 3 \\ 4(x-3)^4, 3 \le x \le 4 \end{cases}$$

We will graph this one subinterval at a time for the equally spaced subintervals of width 1.

```
pow  <- c(1,2, 3, 4)
plot(NULL, xlim=c(0, 4), ylim=c(-5, 5), las=1, ylab="strange function", xlab="x")
plot(function(x) (x - 0)^pow[1], xlim=c(0, 1), add=T)
for(i in 2:length(pow)){
  cat("interval is", 0 + (i-1)*1, "to", 0 + (i)*1,"\n")
  plot(function(x)pow[i]*(x-(0+(i-1)*1))^pow[i], xlim=c(0 + (i-1)*1, 0 + (i)*1), add=T)
}
```




```
## interval is 1 to 2
## interval is 2 to 3
## interval is 3 to 4
```